Exercise 15

Nosky, Christopher

2/17/2021

These assignments are here to provide you with an introduction to the “Data Science” use for these tools. This is your future. It may seem confusing and weird right now but it hopefully seems far less so than earlier in the semester. Attempt these homework assignments. You will not be graded on your answer but on your approach. This should be a, “Where am I on learning this stuff” check. If you can’t get it done, please explain why.

Include all of your answers in a R Markdown report.

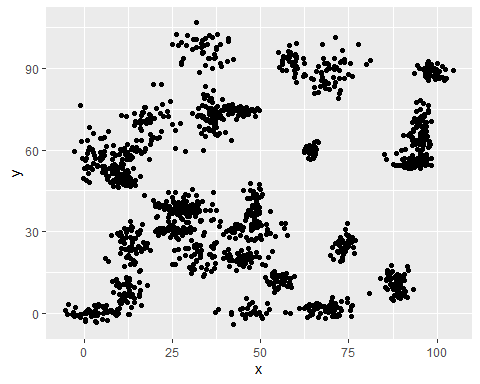
Regression algorithms are used to predict numeric quantity while classification algorithms predict categorical outcomes. A spam filter is an example use case for a classification algorithm. The input dataset is emails labeled as either spam (i.e. junk emails) or ham (i.e. good emails). The classification algorithm uses features extracted from the emails to learn which emails fall into which category.

In this problem, you will use the nearest neighbors algorithm to fit a model on two simplified datasets. The first dataset (found in binary-classifier-data.csv) contains three variables; label, x, and y. The label variable is either 0 or 1 and is the output we want to predict using the x and y variables. The second dataset (found in trinary-classifier-data.csv) is similar to the first dataset except that the label variable can be 0, 1, or 2.

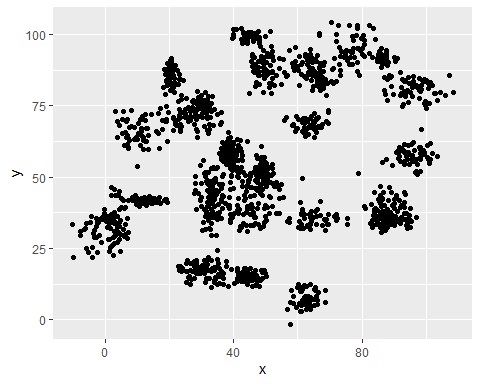
Note that in real-world datasets, your labels are usually not numbers, but text-based descriptions of the categories (e.g. spam or ham). In practice, you will encode categorical variables into numeric values.

1. Plot the data from each dataset using a scatter plot.

ggplot(binary\_set, aes(x = x, y = y)) + geom\_point()



ggplot(trinary\_set, aes(x = x, y = y)) + geom\_point()



1. The k nearest neighbors algorithm categorizes an input value by looking at the labels for the k nearest points and assigning a category based on the most common label. In this problem, you will determine which points are nearest by calculating the Euclidean distance between two points. As a refresher, the Euclidean distance between two points:

normalize <- function(x)   
 return ((x - min(x)) / (max(x) - min(x)))  
  
binary\_set\_norm <- as.data.frame(lapply(binary\_set[,1:3], normalize))  
head(binary\_set\_norm)

## label x y  
## 1 0 0.6930933 0.7861557  
## 2 0 0.7303243 0.8290011  
## 3 0 0.7194983 0.8675355  
## 4 0 0.6523084 0.7674851  
## 5 0 0.6765989 0.7984207  
## 6 0 0.7054045 0.8150698

dist\_binary <- dist(binary\_set\_norm, method = 'euclidean')  
head(dist\_binary)

## [1] 0.05676156 0.08555637 0.04485535 0.02055471 0.03142595 0.05909748

normalize <- function(x)   
 return ((x - min(x)) / (max(x) - min(x)))  
  
trinary\_set\_norm <- as.data.frame(lapply(trinary\_set[,1:3], normalize))  
head(trinary\_set\_norm)

## label x y  
## 1 0 0.3395560 0.3890216  
## 2 0 0.3495902 0.5037685  
## 3 0 0.3735363 0.4801528  
## 4 0 0.3605825 0.4041591  
## 5 0 0.3779910 0.4442837  
## 6 0 0.3708747 0.4326334

dist\_trinary <- dist(trinary\_set, method = 'euclidean')  
head(dist\_trinary)

## [1] 12.202556 10.455806 2.967887 7.420393 5.928894 14.175974

Fitting a model is when you use the input data to create a predictive model. There are various metrics you can use to determine how well your model fits the data. You will learn more about these metrics in later lessons. For this problem, you will focus on a single metric; accuracy. Accuracy is simply the percentage of how often the model predicts the correct result. If the model always predicts the correct result, it is 100% accurate. If the model always predicts the incorrect result, it is 0% accurate.

* There are a couple methods we can use to check the accuracy of our model. First we will split the data sets into training and test sets. We’ll start with the binary data set followed by the trinary set.

set.seed(123) #Getting random sample  
dat.d <- sample(1:nrow(binary\_set\_norm), size=nrow(binary\_set)\*0.7, replace = FALSE) # Selects 70% of data at random  
  
train.binary <- binary\_set[dat.d,] # 70% training data  
test.binary <- binary\_set[-dat.d,] # remaining 30% for testing  
  
# Now creating seperate dataframe for 'Creditability' feature which is our target  
train.binary\_labs <- binary\_set[dat.d,1]  
test.binary\_labs <- binary\_set[-dat.d, 1]  
  
NROW(train.binary\_labs)

## [1] 1048

bin\_knn\_3 <- knn(train=train.binary, test=test.binary, cl=train.binary\_labs, k=3)

* Now that the data is split we can check accuracy. Below you’ll see the accuracy calculation for K = 3 along with a Confusion matrix that will also show us the accuracy.

# Let's calculate the proprotion of correct classification for k = 3  
  
bin\_acc\_3 <- 100 \* sum(test.binary\_labs == bin\_knn\_3)/NROW(test.binary\_labs) # For knn = 3  
bin\_acc\_3 # Accuracy

## [1] 98

# ConfusionMatrix  
confusionMatrix(table(bin\_knn\_3, test.binary\_labs))

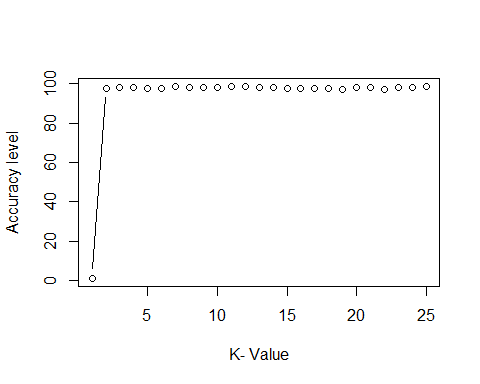
## Confusion Matrix and Statistics  
##   
## test.binary\_labs  
## bin\_knn\_3 0 1  
## 0 227 5  
## 1 4 214  
##   
## Accuracy : 0.98   
## 95% CI : (0.9624, 0.9908)  
## No Information Rate : 0.5133   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.96   
##   
## Mcnemar's Test P-Value : 1   
##   
## Sensitivity : 0.9827   
## Specificity : 0.9772   
## Pos Pred Value : 0.9784   
## Neg Pred Value : 0.9817   
## Prevalence : 0.5133   
## Detection Rate : 0.5044   
## Detection Prevalence : 0.5156   
## Balanced Accuracy : 0.9799   
##   
## 'Positive' Class : 0   
##

* We can also run a calculation showing the accuracy metric for multiple values of K.

i=1 # Declaration to initiate for loop  
bin.k.optm=1 # Declaration to initiate for loop  
for (i in 2:25){  
 bin.knn.mod <- knn(train=train.binary, test=test.binary, cl=train.binary\_labs, k=i)  
 bin.k.optm[i] <- 100 \* sum(test.binary\_labs == bin.knn.mod)/NROW(test.binary\_labs)  
 k=i  
 cat(k,'=', bin.k.optm[i], '\n')  
}

## 2 = 97.77778   
## 3 = 98   
## 4 = 98.22222   
## 5 = 97.55556   
## 6 = 97.77778   
## 7 = 98.44444   
## 8 = 98.22222   
## 9 = 98.22222   
## 10 = 98.22222   
## 11 = 98.66667   
## 12 = 98.66667   
## 13 = 98.22222   
## 14 = 98   
## 15 = 97.55556   
## 16 = 97.77778   
## 17 = 97.55556   
## 18 = 97.55556   
## 19 = 97.33333   
## 20 = 98   
## 21 = 98.22222   
## 22 = 97.33333   
## 23 = 98.22222   
## 24 = 98.22222   
## 25 = 98.44444

plot(bin.k.optm, type='b', xlab='K- Value', ylab='Accuracy level')



* It would appear that K values 2-25 would all work well with the highest percentage coming in with K = 10 & K = 11. Next I’ll do the same thing for the Trinary data set.

# Normalize the data  
normalize <- function(x)   
 return ((x - min(x)) / (max(x) - min(x)))  
  
trinary\_set\_norm <- as.data.frame(lapply(trinary\_set[,1:3], normalize))  
head(trinary\_set\_norm)

## label x y  
## 1 0 0.3395560 0.3890216  
## 2 0 0.3495902 0.5037685  
## 3 0 0.3735363 0.4801528  
## 4 0 0.3605825 0.4041591  
## 5 0 0.3779910 0.4442837  
## 6 0 0.3708747 0.4326334

# Split for training / test  
  
set.seed(123) #Getting random sample  
dat.d <- sample(1:nrow(trinary\_set\_norm), size=nrow(trinary\_set)\*0.7, replace = FALSE) # Selects 70% of data at random  
  
train.trinary <- trinary\_set[dat.d,] # 70% training data  
test.trinary <- trinary\_set[-dat.d,] # remaining 30% for testing  
  
# Now creating seperate dataframe.  
train.trinary\_labs <- trinary\_set[dat.d,1]  
test.trinary\_labs <- trinary\_set[-dat.d, 1]  
  
NROW(train.trinary\_labs)

## [1] 1097

tri\_knn\_3 <- knn(train=train.trinary, test=test.trinary, cl=train.trinary\_labs, k=3)  
  
# Let's calculate the proprotion of correct classification for k = 3  
  
tri\_acc\_3 <- 100 \* sum(test.trinary\_labs == tri\_knn\_3)/NROW(test.trinary\_labs) # For knn = 3  
tri\_acc\_3 # Accuracy

## [1] 93.20594

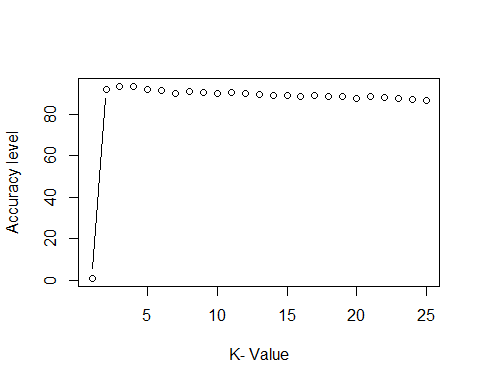
# Confusion Matrix  
confusionMatrix(table(tri\_knn\_3, test.trinary\_labs))

## Confusion Matrix and Statistics  
##   
## test.trinary\_labs  
## tri\_knn\_3 0 1 2  
## 0 127 6 2  
## 1 7 182 11  
## 2 0 6 130  
##   
## Overall Statistics  
##   
## Accuracy : 0.9321   
## 95% CI : (0.9054, 0.9531)  
## No Information Rate : 0.4119   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.8964   
##   
## Mcnemar's Test P-Value : 0.3147   
##   
## Statistics by Class:  
##   
## Class: 0 Class: 1 Class: 2  
## Sensitivity 0.9478 0.9381 0.9091  
## Specificity 0.9763 0.9350 0.9817  
## Pos Pred Value 0.9407 0.9100 0.9559  
## Neg Pred Value 0.9792 0.9557 0.9612  
## Prevalence 0.2845 0.4119 0.3036  
## Detection Rate 0.2696 0.3864 0.2760  
## Detection Prevalence 0.2866 0.4246 0.2887  
## Balanced Accuracy 0.9620 0.9366 0.9454

i=1 # Declaration to initiate for loop  
tri.k.optm=1 # Declaration to initiate for loop  
for (i in 2:25){  
 tri.knn.mod <- knn(train=train.trinary, test=test.trinary, cl=train.trinary\_labs, k=i)  
 tri.k.optm[i] <- 100 \* sum(test.trinary\_labs == tri.knn.mod)/NROW(test.trinary\_labs)  
 k=i  
 cat(k,'=', tri.k.optm[i], '\n')  
}

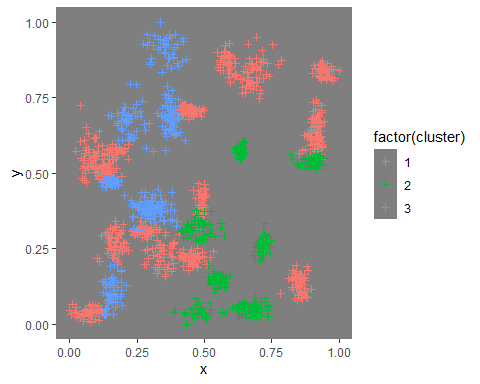
## 2 = 92.14437   
## 3 = 93.20594   
## 4 = 93.41826   
## 5 = 92.14437   
## 6 = 91.29512   
## 7 = 90.23355   
## 8 = 90.87049   
## 9 = 90.44586   
## 10 = 90.02123   
## 11 = 90.44586   
## 12 = 89.80892   
## 13 = 89.38429   
## 14 = 88.95966   
## 15 = 89.17197   
## 16 = 88.74735   
## 17 = 88.95966   
## 18 = 88.53503   
## 19 = 88.74735   
## 20 = 87.68577   
## 21 = 88.53503   
## 22 = 88.1104   
## 23 = 87.68577   
## 24 = 87.26115   
## 25 = 86.83652

plot(tri.k.optm, type='b', xlab='K- Value', ylab='Accuracy level')

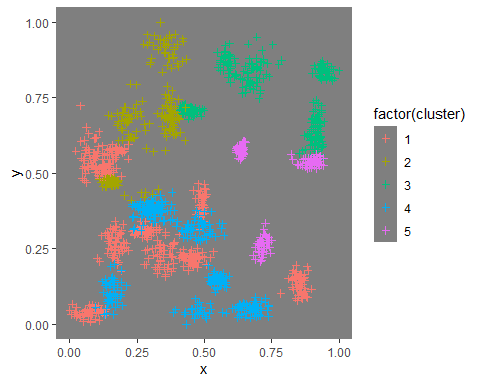


Fit a k nearest neighbors model for each dataset for k=3, k=5, k=10, k=15, k=20, and k=25. Compute the accuracy of the resulting models for each value of k. Plot the results in a graph where the x-axis is the different values of k and the y-axis is the accuracy of the model.

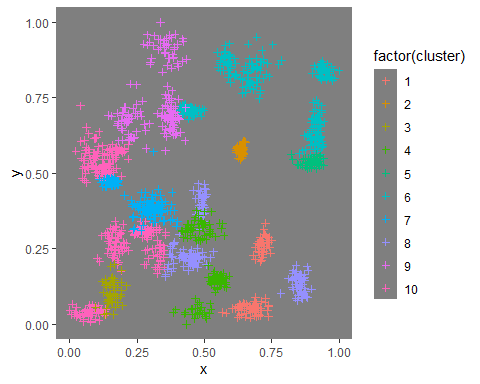
# Build a kmeans model for the Binary data set  
  
# Build a kmeans model  
bin\_km\_3 <- kmeans(binary\_set\_norm, centers = 3)  
bin\_km\_5 <- kmeans(binary\_set\_norm, centers = 5)  
bin\_km\_10 <- kmeans(binary\_set\_norm, centers = 10)  
bin\_km\_15 <- kmeans(binary\_set\_norm, centers = 15)  
bin\_km\_20 <- kmeans(binary\_set\_norm, centers = 20)  
bin\_km\_25 <- kmeans(binary\_set\_norm, centers = 25)  
  
# Extract the cluster assignment vector from the kmeans model  
clust\_bin\_km\_3 <- bin\_km\_3$cluster  
clust\_bin\_km\_5 <- bin\_km\_5$cluster  
clust\_bin\_km\_10 <- bin\_km\_10$cluster  
clust\_bin\_km\_15 <- bin\_km\_15$cluster  
clust\_bin\_km\_20 <- bin\_km\_20$cluster  
clust\_bin\_km\_25 <- bin\_km\_25$cluster  
  
# Create a new data frame appending the cluster assignment  
binary\_set\_km\_3 <- mutate(binary\_set\_norm, cluster = clust\_bin\_km\_3)  
binary\_set\_km\_5 <- mutate(binary\_set\_norm, cluster = clust\_bin\_km\_5)  
binary\_set\_km\_10 <- mutate(binary\_set\_norm, cluster = clust\_bin\_km\_10)  
binary\_set\_km\_15 <- mutate(binary\_set\_norm, cluster = clust\_bin\_km\_15)  
binary\_set\_km\_20 <- mutate(binary\_set\_norm, cluster = clust\_bin\_km\_20)  
binary\_set\_km\_25 <- mutate(binary\_set\_norm, cluster = clust\_bin\_km\_25)  
  
  
## Plotting K = 3,5,10,15,20,25  
  
# Plot the positions of the players and color them using their cluster  
ggplot(binary\_set\_km\_3, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



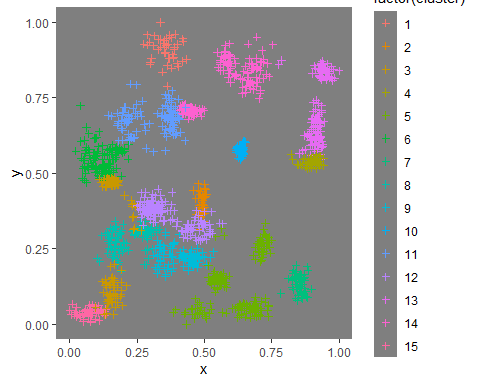
ggplot(binary\_set\_km\_5, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



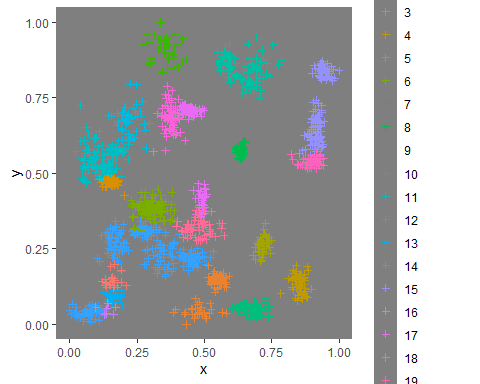
ggplot(binary\_set\_km\_10, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



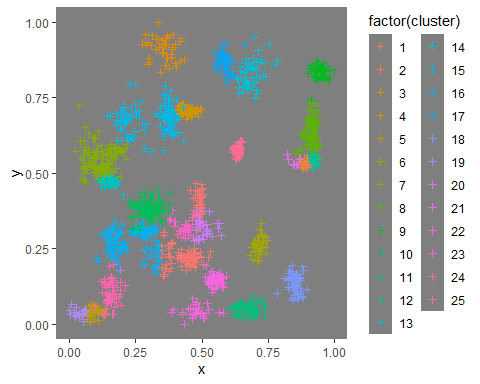
ggplot(binary\_set\_km\_15, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



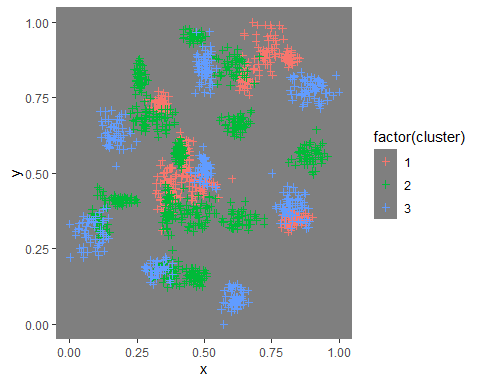
ggplot(binary\_set\_km\_20, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



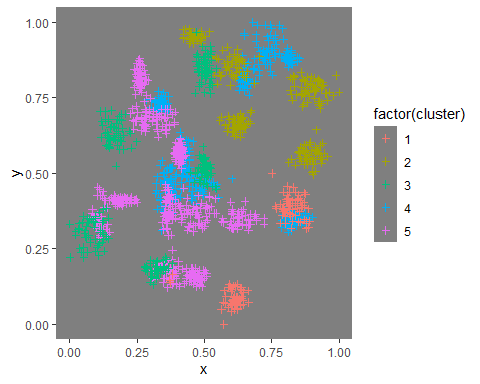
ggplot(binary\_set\_km\_25, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



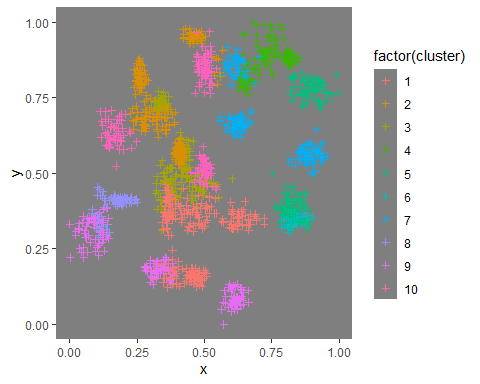
# Build a kmeans model for the Trinary data set  
  
tri\_km\_3 <- kmeans(trinary\_set\_norm, centers = 3)  
tri\_km\_5 <- kmeans(trinary\_set\_norm, centers = 5)  
tri\_km\_10 <- kmeans(trinary\_set\_norm, centers = 10)  
tri\_km\_15 <- kmeans(trinary\_set\_norm, centers = 15)  
tri\_km\_20 <- kmeans(trinary\_set\_norm, centers = 20)  
tri\_km\_25 <- kmeans(trinary\_set\_norm, centers = 25)  
  
# Extract the cluster assignment vector from the kmeans model  
clust\_tri\_km\_3 <- tri\_km\_3$cluster  
clust\_tri\_km\_5 <- tri\_km\_5$cluster  
clust\_tri\_km\_10 <- tri\_km\_10$cluster  
clust\_tri\_km\_15 <- tri\_km\_15$cluster  
clust\_tri\_km\_20 <- tri\_km\_20$cluster  
clust\_tri\_km\_25 <- tri\_km\_25$cluster  
  
# Create a new data frame appending the cluster assignment  
trinary\_set\_km\_3 <- mutate(trinary\_set\_norm, cluster = clust\_tri\_km\_3)  
trinary\_set\_km\_5 <- mutate(trinary\_set\_norm, cluster = clust\_tri\_km\_5)  
trinary\_set\_km\_10 <- mutate(trinary\_set\_norm, cluster = clust\_tri\_km\_10)  
trinary\_set\_km\_15 <- mutate(trinary\_set\_norm, cluster = clust\_tri\_km\_15)  
trinary\_set\_km\_20 <- mutate(trinary\_set\_norm, cluster = clust\_tri\_km\_20)  
trinary\_set\_km\_25 <- mutate(trinary\_set\_norm, cluster = clust\_tri\_km\_25)  
  
  
## Plotting K = 3,5,10,15,20,25  
  
ggplot(trinary\_set\_km\_3, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



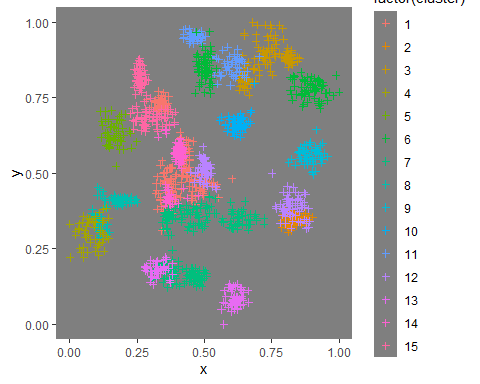
ggplot(trinary\_set\_km\_5, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



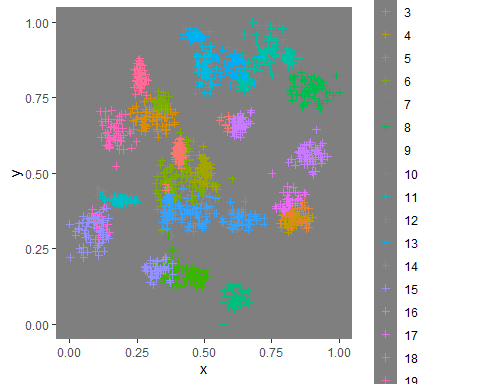
ggplot(trinary\_set\_km\_10, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



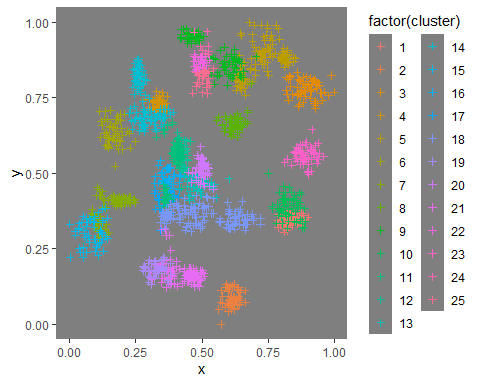
ggplot(trinary\_set\_km\_15, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



ggplot(trinary\_set\_km\_20, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



ggplot(trinary\_set\_km\_25, aes(x = x, y = y, color = factor(cluster))) +  
 geom\_point(shape=3) + theme\_dark() + theme(panel.grid=element\_blank())



1. In later lessons, you will learn about linear classifiers. These algorithms work by defining a decision boundary that separates the different categories.

Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?

* I think it might work well, though to be honest I am only guessing at this point and I don’t have enough insight to know yet.